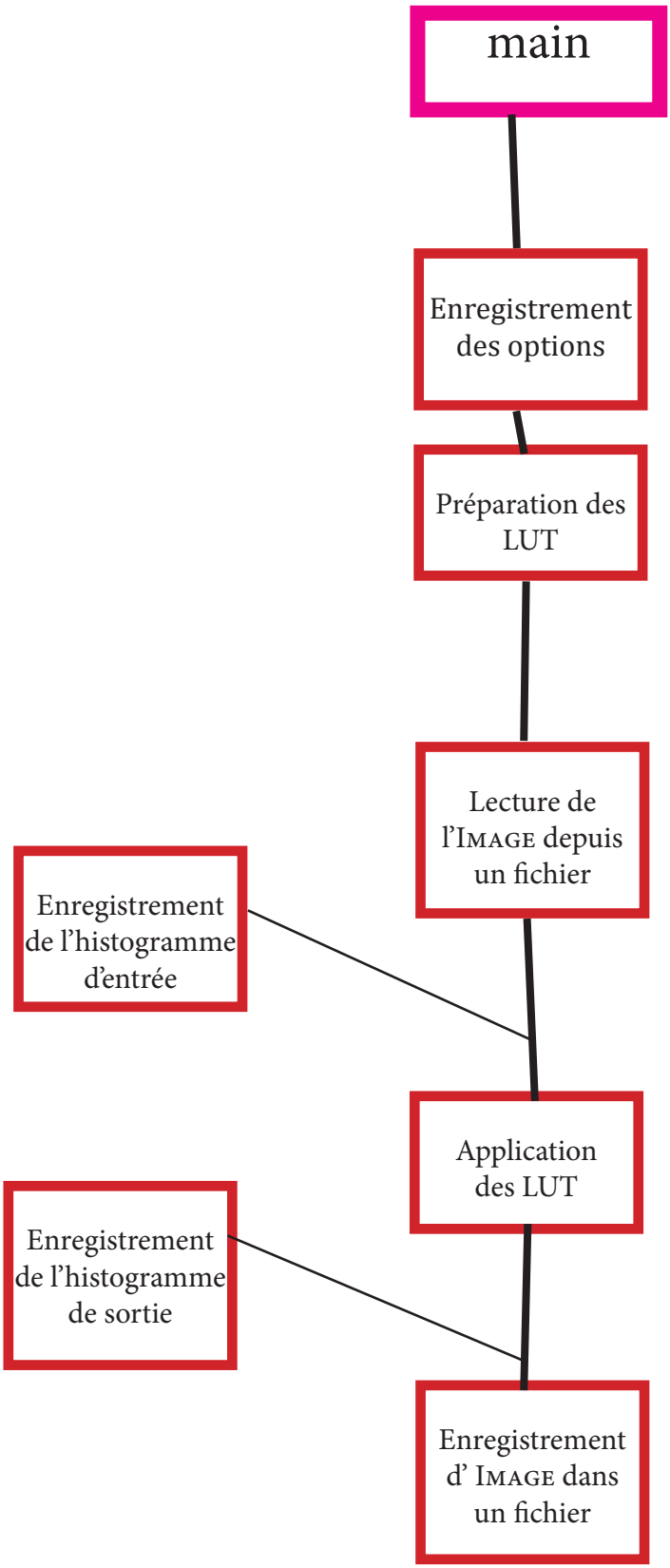
THIEL Pierre
VALLET Gaëlle

IMAC 1

Programmation-Algotrithmique :
Projet de Programation C
Semestre 1
-
Traitement d’images

	Implémenté	Fonctionnel
Structure d'image contenant trois tableaux (canaux pour chaque couleur), RGB et NDG	X	X
Vecteur contenant les LUT	X	X
Ouvrir charger et lire un fichier .ppm	X	X
Ecrire et sauvegarder un fichier .ppm	X	X
Lecture des paramètres entrés par l'utilisateur	X	X
LUT : Augmentation de la luminosité	X	X
Baisse de la luminosité	X	X
Augmentation du contaste	X	X
Baisse du contraste	X	X
Inversion des couleurs	X	X
Effet Sépia	X	X
Flou linéraire naïf	X	X
Flou linéaire récursif	X	
Niveau de gris	X	X
Enveloppes : Ellipse intérieure et extérieure	X	X

Déroulement du programme



De par la nature du format PPM, le choix le plus pertinent de traitement des données du fichier nous a semblé être :
-l'ouverture et l'écriture des données images en tant que fichier binaire.
- l'ouverture et l'écriture des métadonnées de l'en-tête en texte.
Ainsi, nous ouvrons et écrivons les fichiers en mode binaire mais nous les interprétons à certains moments comme des fichiers textes.

La fonction principale, le *main* :

Notre objectif est de rendre transparentes les étapes du programme à la seule lecture du main. Ainsi, toutes les ouvertures et fermetures des fichiers sont ordonnées par celui-ci. Il ne manipule qu'un seul argument **argv[0]** (chemin du fichier source), brièvement, pour vérifier qu'il existe et enregistrer le chemin du fichier, avant que les fonctions de getopt.h ne change l'ordre des arguments.

Dans cette même logique, le *main* appelle les fonctions générales de chaques étape pour assurer le déroulement logique du programme : ainsi, il commence par faire analyser les arguments et options de ARGV par *attraperOptions*, puis ordonne la création du vecteur des LUT accordément, avant d'ouvrir le fichier source et de le charger dans la structure IMAGE.

Un point sur la structure IMAGE :

Notre choix de structure pour les données image est motivé par la lisibilité et l'efficacité de traitement. En effet, il décompose les données brutes de l'images dans les trois canaux ROUGE, VERT et BLEU, tous interprétés comme des tableaux dynamiques de caractères non signés. Nous aurions pu les structurer sous forme de pixels, mais le traitement de l'image par canaux que nous implémentons correspond mieux à notre idée. Les deux autres champs importants de cette structure sont la LARGEUR et HAUTEUR de l'image, dont la connaissance nous est indispensable.

Les champs restants sont plus anecdotiques : un champ de format de couleur (RVB ou NDG), dont nous aurions pu nous passer, permet dans le cas où l'on a applati l'image en niveau de gris avec la LUT **NOIREETBLANC**, de se restrindre à un seul canal, et ainsi diviser le temps traitement par trois. Un ajout de la moyenne de luminosité pour chaque canal, sous la forme d'un vecteur à trois entrées, permet de ne pas s'embarasser à les copier jusqu'au calcul éventuel de contraste avec **DIMCON** et **ADDCON**. Il aurait aussi permis la mise à jour de ces moyennes après l'application de chaque LUT, que nous n'avons implémentée : si son calcul est aisé dans le cas d'un **ADDLUM** ou **DIMLUM**, connaître une valeur précise dans le cas d'une enveloppe est difficile.

Nous allons ainsi de la mémoire dynamiquement pour chacun des canaux de l'image, mais aussi pour l'image elle-même : cela nous permet de ne manipuler que des pointeurs et rend les arguments des fonctions plus homogènes.

Lecture et écriture d'une IMAGE :

La lecture d'une structure IMAGE à partir d'un Fichier est détaillée dans la fonction *lireImage* de image.c.
On commence par lire l'entête, en enregistrant la largeur et la hauteur de l'image, tout en vérifiant sa conformité à la contrainte de l'exercice (le format P6 PNB). On ignore tout commentaire en appelant *lireCommentaire* qui se contente de passer le texte du commentaire jusqu'à une nouvelle ligne. Comme on l'a mentionné tout à l'heure, on utilise ici les fonctions *fscanf* et *fgetc* (bibliothèque stdio.h) de lecture texte car elles sont adaptées à la syntaxe. Les informations recueuillies par *lireEntete* nous permettent de créer un élément de la structure IMAGE adapté avec *creerImage*.

Vient ensuite l'enregistrement des valeurs de luminosité des cellules de l'image dans les canaux. Le fichier est cette fois lut en binaire, et les données sont distribuées aux trois canaux. On calcule en parallèle leurs moyennes de luminosité. On peut ensuite renvoyer au MAIN un pointeur vers la structure IMAGE que l'on vient d'allouer.

Note : *Ce calcul de moyenne est fait récursivement : la fonction sur flottants* calculMoyenne *de fonctions_annexes.c prend en argument la moyenne précédente, la pondère puis lui ajoute un nouveau terme. Cette méthode permet d'éviter tout dépassement lié au type de* caractère non signé*, au prix d'une perte négligeable de précision sur le résultat final lors de sa reconversion en caractère.*

L'écriture d'une structure Image sur un Fichier est détaillée dans la fonction *ecrireImage* de image.c.
Son fonctionnement est analogue à *lireImage*. Notons cependant que dans le cas d'une Image de format NDG, on écrira de façon redondante les mêmes informations trois fois sur le fichier de destination. En effet, s'il existe un format BPM en niveau de gris, il est préférable de l'enregistrer en P6 de sorte qu'on puisse rééditer l'image avec ce même programme.

Construction de l'histogramme :

L'option **[—histo]**, si rentrée par l'utilisateur, est récupérée par *main*, qui ouvre le cas échéant un fichier de chemin défini dans *generalRenommer*, puis passe la structure Image à la fonction principale de histo.c, *lireHisto*, qui déduit la structure Image de l'histogramme de celle-ci, avant de l'enregistrer comme une image en appellant *ecrireImage*.

La structure Histogramme :

On détaille le procédé dans histo.c. On va en réalité passer par la structure Histogramme intermédiaire. Cette structure partage le format de couleur de l'Image originale (utilisé si elle est en niveau de gris). Sa largeur et sa hauteur sont décidées en pré-compilation, mais par souci de modularité, on les garde en tant que champs. lumMax correspond au plus grand nombre de cellules de l'image qui partagent le même niveau de luminosité : cela nous permettra d'adapter l'histogramme à un cadre fixe, en me normalisant. Un Histogramme comprend bien sûr aussi une matrice de valeurs, qui correspond au nombre de cellulles à un certain niveau de luminosité, pour chaque canal. Ainsi, lumMax est le maximum de cette matrice. On notera que lumMax est typé en *entier long*, de sorte que la valeur ne puisse pas déborder (au maximum le nombre de pixels).

Comme ces luminosités calculées sont par canal, l'histogramme produit contiendra les trois composantes Rouge, Verte et Bleue séparées.

Comme la structure Image, on préfère ne manipuler que des pointeurs vers la structure Histogramme, quitte à allouer dynamiquement de l'espace. La première étape intéressante est le remplissage de l'histogramme par *lireCanauxEtLumPourHisto* : dans le cas du format RVB, on parcourt l'image source par canaux et compte le nombre de cellules à la luminosité v, pour incrémenter à chaque fois la v-ième valeur de l'histogramme, et on en profite pour déterminer lumMax. Si le format est RVB, on se restreint à un seul canal, et la structure Histogramme hérite du format de la structure Image.

La seconde étape intéressante est le mouvement inverse : de la structure Histogramme à un histogramme en structure Image, avec *histoVersImage*. On parcourt par lignes et
colonnes la structure Image que l'on veut remplir.
Il est essentiel de normaliser la hauteur des pics en divisant par lumMax, afin de garder une hauteur d'image histogramme fixe.
Pour un canal donné, si la c-ième valeur de luminosité est l, on remplit par VAL_MAX les cellules de colonne c à partir de ligne l, jusqu'à 0, ce qui nous donne un pic. Une fois
l'histogramme de structure Image achevé, on renvoie le pointeur au main.

Un mot sur la structure des LUT et Enveloppes:

Nous définissons une LUT comme une tranformation à appliquer à un point d'une structure Image, qui dépendra d'un certain nombre de paramètres qui constituent un de ses champs. Ces paramètres peuvent eux-mêmes être modulés par une structure d'Enveloppe, au fonctionnement similaire, et dont le rôle est celui d'opérateur sur la LUT dont elle est un champ. LUT et Enveloppes sont identifiées à l'aide d'énumarations, pour rendre leur lecture plus intuitive.
De manière générale, une enveloppe permet de changer l'intensité d'une transformation suivant la position d'un point sur l'image : par exemple, l'effet **DIMLUM-ELLIPSE** assombrit les bords de l'image.
Nous verrons bientôt que toutes les LUT ne sont pas gérées de la même façon. On dira que les LUT **FLOU** et **FLOU2** se comportent comme des ***Planches***, et toutes les autres comme des ***Piles***.

La préparation des LUT :

Les instructions quant à la gestion des LUT se trouve dans gestion_lut.c. La première fonction appelée par *main*, *preparationLUT*, est une fonction principale qui ordonne le calcul du tableau dynamique qui nous servira à contenir les informations sur les transformations demandées. Elle contient des informations techniques sur chaque LUT et Enveloppe, et tout ajout de nouvelle LUT au programme requiert une mise à jour des éléments déclarés dans cette fonction. Par exemple, rajouter une LUT **PIXELLISER** nécessitera, en plus de la rajouter à l'énumération dans la structure :

-Incrémenter le nombre de choix de LUT;
-Rajouter la chaîne de caractère "PIXELLISER") à choixLUT, **en s'assurant que l'indice de cette chaîne correspond à l'indice de PIXELLISER dans l'énumération**.
- Rajouter le paramètres requis pour cette LUT au tableau nbParametresLUT, au même indice que choixLUT (et a fortiori l'énumération).
- Si cette LUT est de comportement ***Planche***, il faut l'ajouter à la liste de *generalLUT* et incrémenter le nombre de LUT ***Planches***.

Ainsi, *preparationLUT* appelle *compterLUT* qui permet de déterminer la taille mémoire à allouer ensuite à vectLUT, le tableau dynamique qui contient les LUT. Pour cela, la fonction récupère argv, et compare chacune de ses chaînes aux chaînes de choixLUT avec la fonction *comparerChainesLUT* de fonctions_annexes.c. Si elles se correspondent, on incrémente le nombre de LUT. Une fois terminée, on réitère l'opération avec *creerVectLUT*, en trouvant cette fois la LUT comme ceci : l'indice de la chaîne dans choixLUT donne l'indice de la LUT dans l'énumération car les éléments sont dans le même ordre.
On remarque que *comparerChainesLUT* renvoie **2** si la LUT possède une enveloppe (si le caractère **'-'** est présent après le nom de la LUT). Ainsi, si cette fois la fonction renvoie **2**, on compare la fin de la chaîne de caractère aux éléments de choixEnveloppe et on obtient l'enveloppe par le même procédé que précédemment.

L'application des LUT : Point de vue global

La seconde fonction principale de gestion_lut.c est *generalLUT*. Elle est appelée par *main* une fois que le fichier est chargé dans la structure Image (et que l'histogramme a été écrit, si demandé), et fait appliquer l'ensemble des LUT à toutes les cellules de l' Image. Traitons tout d'abord un cas où aucune LUT n'est de type ***Planche***, ni **NOIRETBLANC**. Dans ce cas, l' Image est au format RVB, on parcourt les canaux et les cellules une seule fois, en appliquant toutes les LUT à un point avant de passer au suivant : on évite d'appliquer une LUT à tous les points, avant de passer la prochaine (ce sera cependant nécessaire dans le cas d'une LUT ***Planche***). On peut remarquer que l'on a créé une copie de notre Image, que l'on met à jours à chaque étape : c'est inutile dans ce cas particulier, mais cela se révèlera obligatoire dans un cas général.
Si maintenant on a appelé **NOIRETBLANC**, on se contente de n'appliquer les LUT que sur un seul canal, le gris (dans l'énumération équivalent au rouge).
On ne peut pas optimiser dans le cas de L'effet **SEPIA**, car l' Image est reconvertie en RVB après être avoir été applatie en NDG.

Dans le cas où vecLUT contient une LUT ***Planche***, on applique toutes les LUT non ***Planches*** qui la précèdent sur chaque élément normalement, puis on l'applique à toutes mes cellules à la suite avant de passer à la prochaine LUT. Cela se traduit dans le code par une boucle qui vérifie à la fin de chaque itération si une nouvelle couche de LUT ***Planche*** doit être appliquée. La raison pour laquelle nous choisissons ce fonctionnement est que ces LUT nécessitent d'avoir appliqué toutes les LUT précédentes à tous les points pour donner un résultat correct : en effet, dans **FLOU** par exemple, on a besoin d'avoir tous les points environnants à jour avant de faire la moyenne de leurs valeurs. De plus, on a ici besoin d'une copie de la structure Image avec ses valeurs prises avant l'application du LUT ***Planche***, car celles-ci permettront de calculer la moyenne à effectivement attribuer au point. Une fois l'opération terminée, on peut libérer la copie Image et vectLUT. Retenons que les LUT sont appliquées dans l'ordre, sauf la conversion en NDG dans le cas **NOIRETBLANC** et **SEPIA** qui est exécutée en amont.

L'application des LUT : point par point

Pour appliquer les LUT point par point, *generalLUT* appelle *appliquerEnveloppe*. Cette dernière applique l'Enveloppe à la LUT si elle en possède une, puis appelle *appliquerLUT* avec en argument la LUT au paramètre d'intensité éventuellement modifié. Cette dernière fonction associe le nom de la LUT à la fonction correspondante, et renvoie la valeur du point modifiée par celle-ci. Chaque fonction restreint les valeurs de retour entre **0** et **VAL_MAX** pour éviter tout débordement.

Les paramètres d'une LUT :

Dans la structure LUT, il y a deux listes de paramètres : ceux de la LUT seule et ceux de la structure Enveloppe associée. En fait, le programme interprétera le premier argument donné à une LUT comme son intensité si elle ne possède pas d'Enveloppe, et celle de l'Enveloppe si oui. Viennent ensuite les argumenets propres à la LUT, puis les arguments propres à l'Enveloppe.
Ainsi, le nombre réel d'arguments à fournir à une LUT est la somme des arguments de la LUT seule et de ceux de son Enveloppe, moins **1**.

Un exemple: ***ADDLUM-ELLIPSE 50 80 90***
ADDLUM *nécessite un paramètre et* ***ELLIPSE*** *trois. On fournit donc 1+3-1 =* ***3*** *arguments. Le premier est l'intensité du gradient qu'appliquera* ***ELLIPSE*** *sur l'intensité d'****ADDLUM****, et les deux autres correspondent aux données de l'ellipse : demi-axe horizontal et demi-axe vertical.*

L'intensité et tous les paramètres de l'Enveloppe sont pensés pour être compris entre **0** et **100**, et le second paramètre propre à la LUT varie selon le cas (ici, seuls **FLOU** et **FLOU2** en possèdent, il correspond au rayon de flou).
Le cas où l'intensité est égale à 0 correspond à la non modification du point.

Note : S'il manque un paramètre, le programme renvoie un message d'erreur. Tout paramètre en trop est ignoré.

Les LUT implémentées :

ADDLUM et DIMLUM : *lumierePlus* et *lumiereMoins*
Argument notable : aucun
Paramètre: intensité (Pourcentage)
Échelle: linéaire

On augmente (resp. dimininue) la valeur du point de **intensité*VAL_MAX**.En particulier, à **0**, on ne modifie pas l'image, à **100** l'image est un aplat blanc (resp. noir).
Pour essayer : $ ./bin/minigimp ./images/farewell.ppm ADDLUM 60 —histo

ADDCON et DIMCON : *contrastePlus* et *contrasteMoins*
Argument notable: moyenne de luminosité pour le canal traîté
Paramètre: intensité (Pourcentage)
Échelle: racine-quatrième

On augmente (resp. dimininue) la distance entre la luminosité du point et la moyenne pour le canal. De par l'échelle choisie, plus cette distance est grande, moins elle est augmentée (resp. moins elle est grande, plus elle est diminuée).
Pour essayer : $ ./bin/minigimp ./images/farewell.ppm ADDCON 90 —histo

INVERT : *inverser*
Argument notable: aucun
Paramètre: aucun
Échelle: constante

On prend le complémentaire à **VAL_MAX** de la valeur du point, soit **VAL_MAX-valeur_point**.
Pour essayer : $ ./bin/minigimp ./images/farewell.ppm INVERT —histo

NOIRETBLANC : *RVBversNDG* de image.c
Argument notable: aucun
Paramètre: aucun
Échelle: constante

Avant d'effectuer les autres transformations, on convertit l' Image en NDG, en faisant la moyenne de luminosité de chaque cellule sur les trois canaux. Aucune modification n'est appliquée au point à partir de *appliquerLUT*.
Pour essayer : $ ./bin/minigimp ./images/farewell.ppm NOIRETBLANC —histo

SEPIA : *vieillephoto*
Argument notable: aucun
Paramètres : intensité, demi-axe horizontal, demi-axe vertical (voir ADDLUM et ELLIPSE)
Échelle : linéaire (voir ADDLUM et ELLIPSE)

Cet effet est équivalent aux effets **NOIRETBLANC ADDLUM-ELLIPSE** dans un premier temps, puis on multiplie les valeurs de luminosités suivant le canal pour teinter l'image en marron-jaune. On passe ainsi par le NDG avant de changer de teinte. Les coefficients ont été choisis empiriquement : ils sont **{1,3 ; 0,8 ; 0,2}** pour les canaux Rouge, Vert et Bleu respectivement.
Pour essayer : ./bin/minigimp ./images/farewell.ppm SEPIA 80 65 65 —histo

FLOU naïf et FLOU2 itératif: *flou* et *flou2*
Arguments notables : la structure IMAGE et sa copie (pour ses canaux)
Paramètres : intensité (Pourcentage), rayon (Distance aux cellules)
Échelle : Moyenne non pondérée

On fait la moyenne des luminosité des cellules comprises dans un losange de demi-diagonale rayon pour chaque canal. La version naïve est très complexe en temps (complexité suivant carré du rayon) mais ne produit pas d'erreur.
La seconde version se sert d'un résultat précédemment calculé pour déduire le prochain, en soutrayant des termes et en ajoutant d'autres. Elle est plus rapide mais très peu précise : cette imprécision est liée à la conversion du résultat en *caractère non signé*, avant d'être réutilisé en *flottant*. Ils sont de type ***Planche***. On a besoin d'avoir deux images : une qui contient les valeurs non modifiées, et l'autre dans laquelle on enregistre les nouvelles valeurs, car pour le calcul sur un point, on a besoin des valeurs des points adjacents précédant l'application du FLOU.
Pour essayer : ./bin/minigimp ./images/farewell.ppm FLOU 80 10 —histo
./bin/minigimp ./images/farewell.ppm FLOU2 80 10 —histo

On peut comparer le temps d'exécution et la précision.

Les Enveloppes :

ELLIPSE et RELLIPSE : ellipse
Argument notables : versBord , un booléen (qui détermine si l'on est dans le cas ELLIPSE ou RELLIPSE)
oblongue, un autre booléen (qui indique si l'ellipse est plus haute que large)
Paramètres: intensité (Pourcentage) demi-axe horizontal (pourcentage de la moitié de la largeur de l' IMAGE, qui peut constituer un maximum), demi-axe vertical (idem avec la hauteur de l' IMAGE)
Échelle: linéaire

Ces deux enveloppes appliquent un gradient linaire à l'intensité de la LUT qui leur correspond. On calcule pour cela le rayon de l'ellipse paramétrée, que l'on compare à la distance du point au centre de l'image.
Pour essayer :
./bin/minigimp ./images/farewell.ppm ADDCON-RELLIPSE 100 80 90 —histo
./bin/minigimp ./images/farewell.ppm FLOU-ELLIPSE 80 10 30 40 —histo

Note : Les enveloppes ne sont compatibles qu'avec les LUT admettant au moins un argument. Elles sont donc incompatibles avec ***NOIRETBLANC*** *et* ***INVERT****.*
SEPIA *fait intervenir* ***ELLIPSE*** *par défaut.*